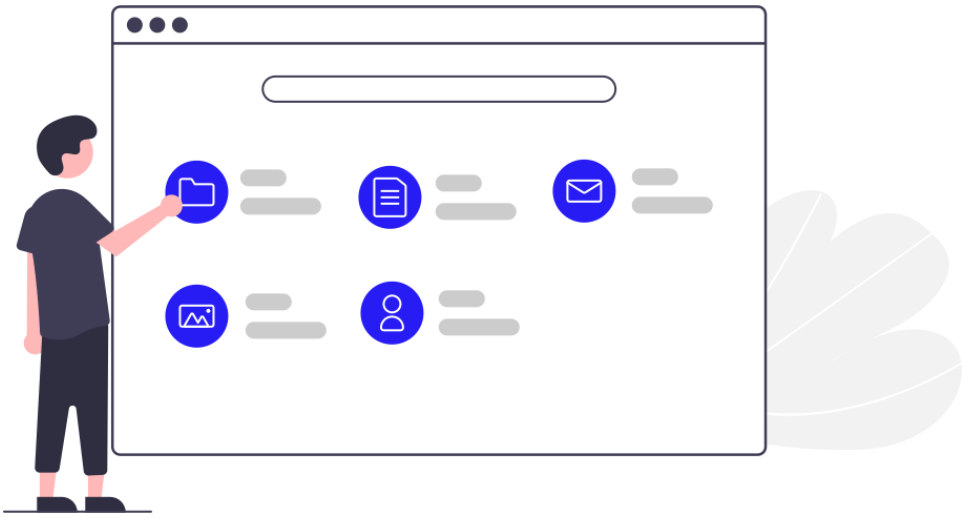
INFO0403

COMPTE RENDU DE TP TOKEN RING



Corentin MACHET
Quentin JUILLIARD
Licence 2e Année, Informatique – URCA

SOMMAIRE

I. INTRODUCTION

1. PREAMBULE

Dans le cadre du module INFO0403 – système d'exploitation – nous nous proposons de réaliser un programme qui simule une architecture donnée.

Il s'agira de mettre en pratique les aspects de la programmation système vus en cours (notamment, gestion des processus, des fichiers, des flux, des signaux, etc.), par le biais d'un réseau en anneau à jeton – « Token Ring » – dont nous calculerons le temps moyen de visite.

En plus du code source, sera fourni un makefile pour la compilation et le présent rapport au format PDF.

NB : pour la compilation, se placer dans le répertoire racine du projet, contenant les sous-répertoires *src*, *obj*, *include*, et *bin*, puis exécuter la commande *make*. Les exécutables sont retournés dans le dossier *bin*.

2. TOKEN RING

L'architecture Token Ring est une topologie principalement utilisée en réseau informatique. Cette technologie se base sur le principe de communication « tour par tour », i.e. dans notre cas, les processus attendent de recevoir le jeton pour continuer à s'exécuter puis passer, à leur tour, ledit jeton au processus suivant.

En ce qui concerne notre projet, un processus père sera chargé d'initialiser un jeton, et de créer *n* processus fils se communicant le jeton donné par le père. A chaque passage du jeton, le processus fils enregistrera le délai de visite de celui-ci dans un fichier. L'ensemble des processus id_i participant à ce système utilisera des tubes et créera un répertoire dans lequel sera stocké les fichiers relatifs au temps de visite.

Il s'agira de récupérer ses données et d'en faire la moyenne par processus, puis la moyenne entre tous les processus fils.

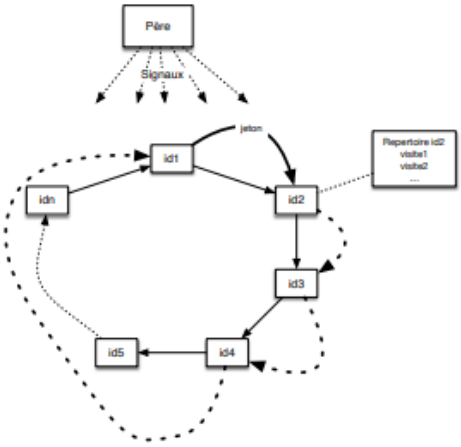


Fig. 1 - Architecture Générale du Système

3. CAHIER DES CHARGES

- Le nombre de processus et le nombre de sauts du jeton sont paramétrables et passés en ligne de commande au processus père.

- Le processus père initialise le jeton, les tubes et les processus fils mais ne participe pas à la circulation du message.
- Chaque processus fils dispose d'un répertoire dans lequel il conserve les fichiers avec les timestamps en microseconde.
- C'est le premier processus fils qui engage la circulation du message.
- Si le père reçoit un signal SIGUSR1, il stoppe (SIGSTOP) ou poursuit (SIGCONT) l'exécution des processus fils.
- Le père termine s'il reçoit un signal SIGQUIT.
- Un autre programme sera chargé de lire les timestamps enregistrés pour calculer la moyenne par processus et la moyenne entre processus du temps de visite du jeton.

4. LANGAGE ET SYSTEME CIBLE

Les programmes seront codés avec le langage C. Les syscalls utilisés seront ceux des API UNIX native et/ou POSIX. Par conséquent, nos sources seront adaptées aux systèmes type Linux notamment.

Le code a été testé sur Ubuntu et Debian.

II. REALISATION

1. STRUCTURES DE DONNEES

La première chose à faire était de gérer les tubes de chaque processus fils. En effet, le père initialise les tubes puis crée ses fils : le contexte du père est donc copié dans celui des fils. Mais chaque fils dispose de tous les tubes ! Il est donc primordial de structurer l'information et d'automatiser l'ouverture et la fermeture des tubes pour chaque processus (sans oublier de libérer la mémoire allouée dynamiquement).

On dispose d'une structure TubeManager qui, à chaque processus, associe les tubes correspondants. Pour automatiser son traitement, nous utiliserons les fonctions suivantes :

- stream_t tubes(int) : alloue dynamiquement un tableau de tube en fonction du nombre de processus demandé par l'utilisateur.
- void createTubeManager(TubeManager*, stream_t**, int, int) : initialise le TubeManager avec les tubes correspondants.
- void closeTubes(stream_t**, int, TubeManager*) : ferme les tubes sauf ceux utilisés par le TubeManager. Si le TubeManager est vide, i.e. NULL, alors tous les tubes sont fermés.
- void freeTubes(stream_t**, int) : désalloue la mémoire du tableau dynamique de tubes créé par la fonction tubes().

Remarque : un processus est caractérisé par son pid.

2. SIGNAUX

Maintenant que les processus fils communiquent, il faut pouvoir gérer la communication avec le processus père.

Pour se faire, nous modifions le gestionnaire de signaux grâce à la fonction signal(), qui renvoie l'action vers la fonction handler(). Selon le signal, le programme réalise une suite d'instructions adaptées : c'est la programmation événementielle.

Ainsi, le père peut, à son tour, envoyer un SIGSTOP ou un SIGCONT à ses fils à l'aide de la fonction kill().

3. FICHIERS

Tout comme pour la gestion des tubes, la sauvegarde des timestamps dans des fichiers nécessitait une série de tests. C'est pourquoi nous avons regroupé trois fonctions dans la source FileManager :

- bool dirExist(int) : vérifie si un répertoire existe ou non dans le répertoire courant.
- void createDir(int) : crée le répertoire associé à l'identifiant passé en paramètre dans le répertoire courant.
- void editTime(int, int, int, int) : édite le timestamp dans le fichier courant. Si le fichier n'existe pas il est créé.

Remarque : les booléens ne sont pas traités nativement en C. Ils sont déclarés dans la source « system ».

Les répertoires portent l'identifiant du processus associés, formaté sur 6 caractères. Ainsi, le répertoire du processus 2 sera nommé *id000002*. Dans chaque répertoire, les fichiers sont nommés selon l'ordre de passage du jeton : par exemple *v000003*.

Le timestamp est directement écrit dans le fichier (encodage int8) : il n'est donc pas lisible au format Plan Text.

III. PRODUCTION

1. DUREE DE VISITE DU TOKEN

Nous réalisons un second programme que nous nommerons « ringstat ».

Celui-ci teste chaque répertoire dans le répertoire courant : s'il correspond à un répertoire de sauvegarde, il y liste les fichiers, récupère les timestamps et en affiche la moyenne. Ensuite ringstat ajoute la moyenne à une variable servant d'accumulateur et réitère avec le dossier suivant. Lorsque le programme termine, il divise la somme de l'accumulateur avec le nombre de répertoires testés pour obtenir la moyenne globale.

2. RESULTATS

On exécute le programme tokenring pour 10 processus et 20 sauts. Puis on récupère les données statistiques grâce au programme ringstat :

```
tehcam@debian:/media/sf_URCA/info0403/bin$ ./tokenring 10 50
Processus pere (pid: 23010) en cours d'execution...
        > Tapez 'kill -s usr1 23010' pour stopper/continuer l'execution des processus f
ils
        > Tapez 'kill -s quit 23010' pour quitter le processus père
================================================================


Fin du processus fils no.0, pid: 23011, token: 0
Fin du processus fils no.1, pid: 23012, token: -1
Fin du processus fils no.2, pid: 23013, token: -2
Fin du processus fils no.3, pid: 23014, token: -3
Fin du processus fils no.4, pid: 23015, token: -4
Fin du processus fils no.5, pid: 23016, token: -5
Fin du processus fils no.6, pid: 23017, token: -6
Fin du processus fils no.7, pid: 23018, token: -7
Fin du processus fils no.8, pid: 23019, token: -8
Fin du processus fils no.9, pid: 23020, token: -9
Terminaison du programme
tehcam@debian:/media/sf_URCA/info0403/bin$
```

```
tehcam@debian:/media/sf_URCA/info0403/bin$ ./ringstat
Temps moyen de visite du processus 0 : 27421.200µs
Temps moyen de visite du processus 1 : 22943.000µs
Temps moyen de visite du processus 2 : 23546.400µs
Temps moyen de visite du processus 3 : 24044.400µs
Temps moyen de visite du processus 4 : 24554.000µs
Temps moyen de visite du processus 5 : 25097.000µs
Temps moyen de visite du processus 6 : 25582.800µs
Temps moyen de visite du processus 7 : 26059.000µs
Temps moyen de visite du processus 8 : 26491.400µs
Temps moyen de visite du processus 9 : 26959.400µs
Temps moyen de visite global des processus : 25269.860µs
tehcam@debian:/media/sf_URCA/info0403/bin$
```

IV. CONCLUSION

Finalement, nous avons pu mettre en place l'architecture Token Ring souhaitée, en respectant les contraintes énoncées dans le cahier de charges.

Le programme adapte le nombre de processus et de sauts selon les paramètres en entrée, et crée un fichier par passage du token. Les flux sont correctement ouverts et fermés, sans fuite mémoire et le processus père gère correctement les signaux qu'il reçoit.

En outre, le deuxième exécutable lit correctement les données enregistrées dans les fichiers des différents répertoires, permettant de calculer les moyennes comme stipulé dans le sujet.